USING REGULAR EXPRESSIONS

NJIT New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

Regular Expressions and Automata

- The matcher is a simple machine called a finite state machine or finite state automata
- Machine has a set of states, a unique start state, a set of end states, a unique end symbol, and transitions from one state to another based on input to the machine
- It's useful to represent these machines as graphs where the vertices are states and the edges are transitions to states

Deterministic FSA

- A finite state automaton is *deterministic* if for each state and each input symbol, there is at most one outgoing arc from that state for each input symbol.
- A Deterministic FSA is a DFA

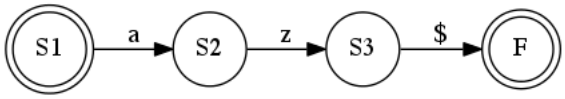
Definitions

- A *configuration* on an fsa consists of a state and the remaining input.
- A *move* consists of traversing the arc exiting the state that corresponds to the leftmost input symbol, thereby consuming it. If no such arc, then:
  - If no input and state is final, then accept.
  - Otherwise, error.

- An input is *accepted* if, starting with the start state, the automaton consumes all the input and halts in a final state.

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Simple example



- FSA to recognize the string “az”
- It is deterministic: there is no ambiguous transition from any one of the states
- Error conditions are not shown; any character other than those labeled on an arc is an error

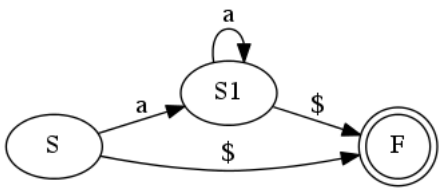
Simple Building Blocks

- Sequence (“abc”)

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

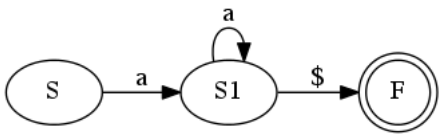
Simple Building Blocks

Zero or more (“a*”)



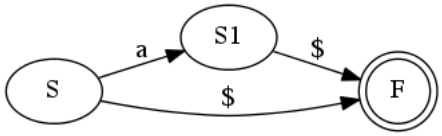
Simple Building Blocks

- One or more (“a+”)



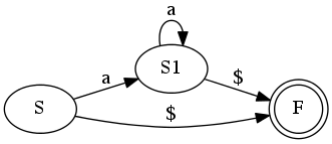
Simple Building Blocks

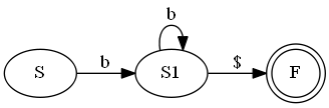
- Zero or one ("a?")

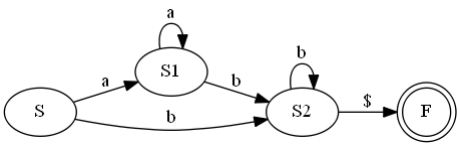


Combinations

- Zero or more a's followed by 1 or more b's (a*b+)

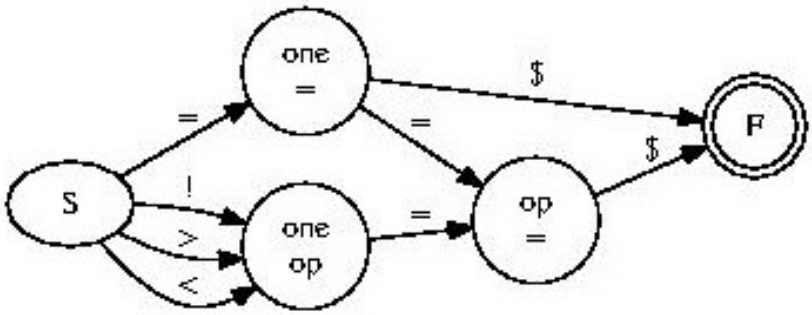






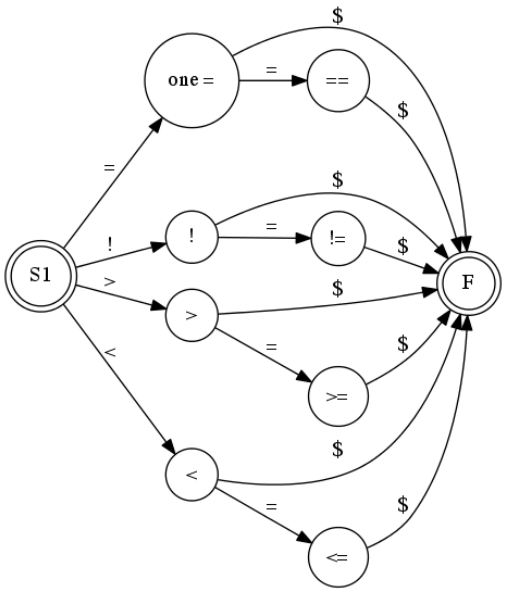
FSA for several operators

- = or == or != or <= or >=
- Expression is: [=!<>]?=



Another FSA for multiple operands

- =, ==, !, !=, >, >=, <, <=
- [=!<>]=?
- Unlike earlier version there's multiple paths through the machine for different operands
- Depending on which $ arc is followed, a different operand is recognized



DFAs in Lexers

- Lexers are machines recognizing multiple patterns at once.
- Instead of “end of input” in a lot of places, they recognize “not part of this pattern” as being the end of the pattern. They then accept the pattern, and go back to the start state
- In an implementation, the character recognized as “not part of this pattern” might be (probably is!) the first character in the next token. The implementation would “push back” or save that character so that it’s the first character processed in the next pattern

Use of a Lexer by a Parser

- Remember, syntactic analysis or parsing needs a stream of tokens
- How to parse?
  - Read entire input, create a complete stream of tokens, match them against the grammar rules?
  - Read a token at a time and match as you read the tokens (this is what’s usually done; it’s easier)
- The Lexer is generally a getToken() function of some kind
- Automatic lexer generators make a function called yylex()

Using Regular Expressions

- Checking for regular expression matches
- Writing regular expressions
- Converting regular expressions to DFAs
- Regex Libraries

Problems: Matching Regular Expressions

- Given a regular expression, does a given string match it?
  - `(a|b)+`
    - `Does aaaababbbab match?`
    - `Does aaaccaaab match?`
  - `[ab]*c?`
    - `Does a match?`
    - `Does bbbcc match?`
    - `Does ac match?`
- NOTE: the suffix (+, *, ?) is a metacharacter that applies to the character/metacharacter/group to its left

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Does this match these strings??

[+-]?[0-9]*\.[0-9]+e[+-][0-9]+

- a floating point number
- "an optional sign, zero or more digits followed by a dot and one or more digits, an e, a sign, and one or more digits"

-3 ← no

.02e+4 ← yes

+6.02e+23 ← yes

5e-2 ← no

Problems: Writing regular expressions

- Regular expression for English proper nouns
  [A-Z][a-z]*

- A binary constant expressed as a zero, a lower- or uppercase b, and a sequence of one or more 0s and 1s
  0[bB][01]+

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Write regular expressions

- Regular expression for words made up of three uppercase letters
- Regular expression for a sequence of words made up of letters and separated by whitespace, and terminated with a punctuation mark. The words can optionally begin with a capital letter. The first word MUST begin with a capital letter

Write regular expressions

- Regular expression for words made up of three uppercase letters

  [A-Z][A-Z][A-Z]

  – Note that there is no notation for counting things, so this is the only way to represent three uppercase letters in sequence

Write regular expressions

- Regular expression for a sequence of words made up of letters and separated by whitespace, and terminated with a punctuation mark. The words can optionally begin with a capital letter. The first word MUST begin with a capital letter

  [A-Z][a-z]*([\t\n]+[A-Za-z][a-z]*)*[\.\?!]

Regex Libraries

- "I have a problem. I will use regular expressions to solve it. Now, I have two problems"

Regex Libraries

- This seems like a great candidate for a package of library routines, doesn't it?
- But there are complexities:
  - differing details of syntax
  - differing implementations
    - API for libraries
    - Thread safety
  - differing versions of libraries and code
    - "boost"
    - <regex>
      - Implementations differed from gcc 4.8 -> 4.9

Regex Library Usage

- The general pattern is to write a regular expression in a string, compile the expression into some object, and to match the compiled expression against a string
- With <regex> when you compile the string into a regular expression, you have to specify which type of encoding you are using
- Several types of encoding (ECMAScript, Basic POSIX, Extended POSIX, awk, grep, egrep)
- Slight variations between encoding rules regarding character classes, handling of spaces, "greedy" or "non-greedy matches", etc.

<regex>

- This library evolved over time and behaves differently between C++ versions
- --std=c++0x and –std=c++11 behave differently
- Different versions of the compiler behave differently

Sample program

```
#include <iostream>
#include <regex>
#include <string>
using namespace std;

int main(int argc, char *argv[])
{
        string s = "Please show me how this works";

        std::regex pattern("[A-Z][a-z]*");

        std::cout << "Ready!" << std::endl;

        if( std::regex_search(s, pattern) ) {
                std::cout << "Search works!" << std::endl;
        }

        if( std::regex_match(s, pattern) ) {
                std::cout << "Matches!" << std::endl;
        }
}
```

Compiles and runs on AFS machines (g++ 4.9.2)
Compiles and fails on Vocareum (g++ 4.8.3)

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES


NJIT
THE EDGE IN KNOWLEDGE